Comunicarea client-server in Visual Basic.NET, utilizand clasa SOCKET

În cadrul acestui laborator, se doreşte construirea a două aplicații, care să comunice utilizând protocolul TCP, una dintre aplicații fiind de tip server, care ascultă cererile ce ajung la ea, iar cealaltă aplicație, jucând rolul unui client, ce inițiaza conexiunea la aplicația server.

Aplicația server poate accepta sau rejecta conexiunea. Dacă serverul acceptă conexiunea, poate începe un dialog între client şi server. Odată ce clientul a terminat ceea ce a avut de făcut, acesta poate închide conexiunea cu serverul. Pe durata unei conexiuni active a unui client la server, clientul poate trimite şi/sau primi date de la server.

De fiecare dată când un partener de comunicație (client sau server) trimite date celuilalt partener se presupune că acesta din urmă recepționează datele respective. Pentru a şti când datele au ajuns la destinație aplicația care primeşte datele are două opțiuni: fie verifică daca datele au ajuns la intervale regulate de timp (polling), fie trebuie sa folosească un fel de mecanism de notificare a momentului în care datele au ajuns la destinație, moment la care aplicația poate citi datele. Deoarece Windows este un sistem de operare care se bazează pe evenimente, sistemul de notificare pare o alegere mai potrivită.

Cele două aplicații se află de obicei pe calculatoare diferite, dar se pot afla de asemenea pe acelaşi calculator.

Pentru a se putea realiza comunicarea între cele două aplicații, trebuie să se stabilească în primul rand o conexiune. Pentru a putea realiza o conexiune, cele două aplicații trebuie să se poată identifica. Acest lucru se face prin intermediul unui IP.

Înainte de dezvoltarea propriu-zisă a aplicației client-server, pentru o mai bună întelegere a lucrurilor, se vor prezenta ***câteva noțiuni generale privind programarea cu socket-uri:***

Protocol -Un *protocol* reprezintă o convenție de reprezentare a datelor folosită în comunicarea între două calculatoare. Având în vedere faptul că orice informație care trebuie trimisă prin rețea trebuie serializată astfel încât să poată fi transmisă secvențial, octet cu octet, către destinație, era nevoie de stabilirea unor convenții (protocoale) care să fie folosite atât de calculatorul care trimite datele cât si de cel care le primeşte.

Cele mai utilizate protocoale sunt **TCP** (Transmission Control Protocol) si **UDP** (User Datagram Protocol):

- *TCP (Transport Control Protocol)* este un protocol ce furnizează un flux sigur de date între două calculatoare. Acest protocol asigură stabilirea unei conexiuni permanente între cele două calculatoare pe parcursul comunicației.
- *UDP (User Datagram Protocol)* este un protocol ce trimite pachete independente de date, numite *datagrame*, de la un calculator către altul fără a garanta în vreun fel ajungerea acestora la destinație. Acest protocol nu stabileşte o conexiune permanentă între cele două calculatoare.

IP – orice calculator gazdă conectat la Internet este identificat în mod unic de adresa sa **IP** (IP este acronimul de la *Internet Protocol*). Aceasta reprezintă un număr reprezentat pe 32 de biti, uzual sub forma a 4 octeți, cum ar fi de exemplu: 193.226.26.231 şi este numit adresa IP *numerică*. Corespunzătoare unei adrese numerice există şi o adresă IP *simbolică*.

Port - Un *port* este un număr de 16 biti care identifică în mod unic procesele care rulează pe o anumită maşină. Orice aplicație care realizează o conexiune în rețea va trebui să ataşeze un număr de port acelei conexiuni. Valorile pe care le poate lua un număr de port sunt cuprinse între 0 şi 65535 (deoarece sunt numere reprezentate pe 16 biti), numerele cuprinse între 0 şi 1023 fiind însă rezervate unor servicii sistem şi, din acest motiv, nu trebuie folosite în aplicații.

Socket - Un *socket (soclu)* este o abstracțiune software folosită pentru a reprezenta fiecare din cele două "capete" ale unei conexiuni între două procese ce rulează într-o rețea. Fiecare socket este ataşat unui port astfel încât să poată identifica unic programul căruia îi sunt destinate datele.

Programarea în rețea în Windows este posibilă prin intermediul socket-urilor. Programarea cu socket-uri se aseamănă cu accesul la fişiere (operațiile pe un socket pot fi asemănate cu operațiile de I/O pe un fişier) sau cu comunicația serială. Socket-urile se pot folosi pentru a construi două aplicații care comunică între ele, una de tip server, iar cealaltă de tip client..

Dezvoltarea aplicației client-server în Visul Basic.NET, utilizând clasa Socket

Interțetele grafice ale aplicațiilor client şi server, vor fi construite după modelele prezentate în Fig.1 şi Fig.2.

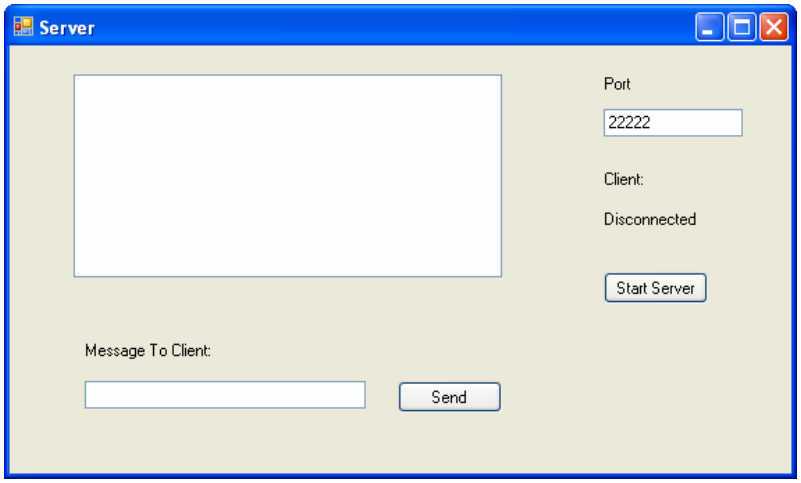

Fig.1 Interfața aplicației server

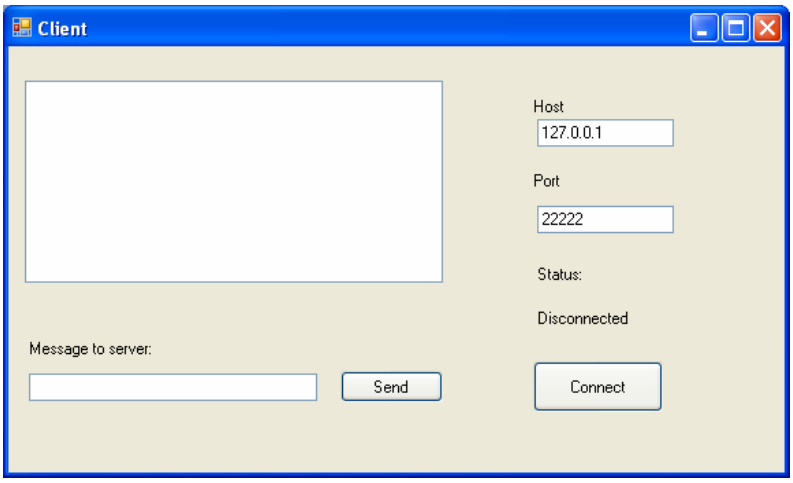

Fig.2 Interfața aplicației client

Programarea cu socket-uri în Visual Basic.NET este posibilă prin utilizarea clasei *Socket.* Clasa **Socket** implementează interfața socketului de tip Berkeley şi se găseşte în **namespace**-ul **System.Net.Sockets.** Clasa Socket furnizează un set bogat de metode şi proprietăți pentru comunicația în rețea. Ea permite atât transferuri de date asincrone cât şi sincrone, folosind oricare dintre protocoalele de comunicație enumerate de ProtocolType.

În cadrul acestei lucrări de laborator, comunicarea între client şi server se face utilizând protocolul TCP, iar conectarea la server şi transferul de date între client şi server se face în mod asincron.

Primul pas în comunicarea dintre cele două aplicații îl constituie pornirea serverului, şi implicit procesul de „ascultare" a serverului pe portul desemnat. Pentru a realiza acest lucru, sunt necesare următoarele:

➢ crearea unui socket asociat serverului, numit socket *listener,* care ascultă un anumit număr de port pentru conexiuni de la clienți (instanțierea unui obiect din clasa Socket):

```
listener = New Socket(AddressFamily.InterNetwork,
SocketType.Stream,ProtocolType.Tcp)
```

Parametrul *addressFamily* specifică schema de adresare folosită de **Socket (**în cazul nostru fiind interNetwork = IP versiunea 4), *socketType* specifică tipul **Socket**-ului, care poate fi *stream* sau d*atagram* (in cazul nostru fiind de tipul *stream*), iar *protocolType* specifică protocolul folosit de **Socket** (în cazul nostru TCP).

➢ crearea unui endpoint pentru server (adresa de IP şi număr de port local) prin instanțierea unui obiect din clasa IPEndPoint (IPAddress address, int port):.

```
Dim ipLocal As New IPEndPoint(IPAddress.Parse("127.0.0.1"),
Convert.ToInt32(_txtBxPort.Text))
```

Clasa IPEndPoint(...) reprezintă un terminal din rețea printr-o adresa de IP şi un număr de port., şi face parte din **namespace**-ul **System.NET.**

➢ socket-ul serverului, mai intâi, va fi asignat unui IP şi port local prin metoda Socket.Bind(endpoint local):

```
listener.Bind(ipLocal)
```

Metoda Bind specifică, practic, endpoint-ul exact cu care va avea loc comunicația.

➢ socketul serverului va trece în starea de ascultare prin apelul Socket.Listen(lungime_coadaasteptare); metoda Listen trebuie apelată după ce socket-ul a fost asigant unui endpoint local:

```
listener.Listen(4)
```

În continuare, se începe o operație asincronă de acceptare a unei încercări de conectare, cu ajutorul metodei **BeginAccept:**

Public Function BeginAccept (callback As AsyncCallback, state As Object) As IAsyncResult

Parametrul *callback* reprezintă un delegat de tip **AsyncCallback** care se va apela în momentul primirii unei cereri de conectare, şi se va executa într-un fir de execuție separat. Parametrul *state* reprezintă un obiect care conține informații de stare referitoare la această cerere.

Parametrul returnat de funcția BeginAccept este de tipul IAsyncResult. IAsyncResult este o interfață care reprezintă starea unei operații asincrone.

Un obiect de tipul IAsyncResult, stochează informații de stare legate de o operație asincronă şi furnizează un obiect de sincronizare, care să permită firelor de execuție să fie anunțate atunci când operația se incheie.

Obiectele de tip IAsyncResult sunt pasate ca şi parametri metodelor invocate de delegatele AsyncCallBack, atunci când o operație asincronă se incheie.

Înainte de apelul metodei **BeginAccept,** va trebui să creem o metodă *callback* care să implementeze delegatul AsyncCallBack. Vom numi această metodă **OnClientConnect.** La apelul metodei **BeginAccept** va trebui să transmitem numele acesteia ca şi parametru**.** Astfel, în momentul în care apare o cerere de conectare din partea unui client se va apela aceasta metoda.

Metoda **OnClientConnect** va trebui să conțină un apel al metodei **EndAccept**, care acceptă în mod asincron o încercare de conectare şi crează un nou **Socket** pentru a face posibilă comunicarea cu calculatorul care a făcut cererea.

Public Function EndAccept(ByVal *asyncResult* As IAsyncResult) As Socket

Metoda **EndAccept** încheie practic, un apel **BeginAccept**.

```
Private Sub OnClientConnect(ByVal asyn As IAsyncResult)
   ..........
   client = listener.EndAccept(asyn)
   ..........
End Sub
```

Când se apelează metoda **BeginAccept**, sistemul va folosi un fir separat de execuție, pentru execuția metodei calback **OnClientConnect**, care se va bloca pe metoda **EndAccept**, până când o conexiune este in asteptare.

```
listener.BeginAccept(New AsyncCallback(AddressOf OnClientConnect), Nothing)
```

```
Imports System.Net
```

```
Imports System.Net.Sockets
Imports System.Text

Public Class ServerForm
  'socket folosit pentru ascultare cereri de conectare
  Private listener As Socket
  Private client As Socket
  .......

  Private Sub _btnStartServer_Click(...)
    ............
    'creez un socket asociat serverului
    listener = New Socket(AddressFamily.InterNetwork,
    SocketType.Stream,ProtocolType.Tcp)
    'se creeaza un endpoint local pentru server
    Dim ipLocal As New IPEndPoint(IPAddress.Parse("127.0.0.1"),
    Convert.ToInt32(_txtBxPort.Text))
    'setez endpoint-ul creat
    listener.Bind(ipLocal)
    'socketul trece in starea de ascultare,
    'coada de asteptare este de 4 clienti;
    listener.Listen(4)
    'incepe operatia asincrona de acceptare a cererilor de conectare
    'orice incercare de conectare va fi tratata de metoda
    'OnClientConnect
    listener.BeginAccept(New AsyncCallback(AddressOf
    OnClientConnect),Nothing)
    ............
  End Sub
  Private Sub OnClientConnect(ByVal asyn As IAsyncResult)
    ...........
    client = listener.EndAccept(asyn)
    ..........
  End Sub
  ............
End Class
```

În acest moment, serverul este pornit, aşteptând o cerere de conectare din partea unui client. Ca urmare, pentru a se putea realiza conexiunea client-server, clientul va trebui să emită o cerere de conectare Pentru aceasta, se efectuează următoarele operații, pe partea de client:

➢ crearea unui socket pentru a face posibila conectarea cu serverul (instanțierea unui obiect din clasa Socket):

```
Dim server As New Socket(AddressFamily.InterNetwork,
socketType.Stream, ProtocolType.Tcp)
```

➢ crearea unui endpoint ce corespunde calculatorului la distanță la care dorim să ne conectăm (server-ului), cu ajutorul clasei IPEndPoint(IPAddress address, int port))

Primul parametru al constructorului trebuie să fie o instanță a clasei IPAddress. Dacă examinăm clasa IPAddress vom vedea că are o metodă statică numită Parse(string) care returnează o instanță a clasei; funcția primeşte ca parametru un obiect de tip string (reprezentarea adresei IP). Al doilea parametru va fi numărul portului.

Dim epServer As New IPEndPoint(IPAddress.Parse(_txtBxHost.Text), Convert.ToInt32(_txtBxPort.Text))

➢ o dată ce avem pregătită instanța clasei IPEndPoint, putem apela metoda **BeginConnect**, care începe o cerere de conectare asincronă la server.

Public Function BeginConnect (remoteEP As EndPoint,callback As AsyncCallback, state As Object) As IAsyncResult

Parametrul *remoteEP* reprezintă endpoint-ul asociat calculatorului cu care se dorește stabilirea conexiunii (în cazul nostru serverul). Parametrul *callback* reprezintă un delegat de tip **AsyncCallback** care va fi apelat de către sistem, cand se inițiaza o cerere de conectare , şi se va executa într-un fir de execuție separat. Parametrul *state* reprezintă un obiect care conține informații de stare referitoare la această cerere. La apel, valoarea parametrului din urmă va fi reprezentată de socket-ul asociat clientului.

Parametrul returnat de funcția BeginAccept este de tipul IAsyncResult.. El va trebui acceptat ca parametru de funcția callback pe care o vom crea.

Înainte de apelul metodei **BeginConnect,** va trebui să creem o metodă *callback* care să implementeze delegatul AsyncCallBack, Vom numi această metodă **ConnectionTry**. La apelul metodei **BeginConnect** va trebui să transmitem numele acesteia ca şi parametru**.** Astfel, în momentul în care apare o cerere de conectare, se va apela aceasta metoda, ea executându-se într-un fir de execuție separat.

```
server.BeginConnect(epServer,New AsyncCallback(AddressOf ConnectionTry),
newsock)
```

Metoda callback creată de noi, va trebui să conțină un apel al metodei **EndConnect**, care încheie cererea de conexiune asincronă, începută în metoda BeginConnect.

Public Sub EndConnect (asyncResult As IAsyncResult)

```
Private Sub ConnectionTry(ByVal iar As IAsyncResult)
   ..........
   server.EndConnect(iar)
   ..........
End Sub
```

Când aplicația noastră, apelează metoda **BeginConnect**, sistemul va folosi un fir separat de execuție, pentru execuția metodei callback **ConnectionTry**, care va rămâne blocat pe metoda **EndConnect**, până când conexiunea la server reuşeşte sau, în caz contrar, până când este captată o excepție.

```
Imports System.Net
Imports System.Net.Sockets
Imports System.Text

Public Class ClientForm
  ..........
  Private Sub _btnConnect_Click(ByVal sender As System.Object, ByVal e As
  System.EventArgs) Handles _btnConnect.Click
     ..........
     ' crearea unui nou socket, pentru conectarea cu serverul
     Dim server As New Socket(AddressFamily.InterNetwork,
     SocketType.Stream, ProtocolType.Tcp)
```

```
        'se creeaza un endpoint pentru server
        Dim epServer As New IPEndPoint(IPAddress.Parse(_txtBxHost.Text),
        Convert.ToInt32(_txtBxPort.Text))
        'se incepe incercarea de stabilire a unei conexiuni
        server.BeginConnect(epServer, New AsyncCallback(AddressOf
        ConnectionSuccessfull), client)
       ...........
    End Sub
    Private Sub ConnectionTry(ByVal iar As IAsyncResult)
      ...........
      server.EndConnect(iar)
      ...........
    End Sub
    ...........
End Class
```

Pornirea serverului şi conectarea unui client la server, vor fi semnalate prin intermediul unor mesaje în interfața grafică a celor două aplicații, după cum se poate vedea în Fig.3 si Fig.4.

Datorită faptului că metodele OnClientConnect şi ConnectionTry create de noi implementează delegate-uri de tip AsyncCallback, rulând pe alte fire de execuție, orice modificare efectuată în cadrul lor asupra controalelor de pe form, se face în felul următor:

```
Private Sub successCon()
  _lblConnStatus.Text = "Connected to: " + server.RemoteEndPoint.ToString()
End Sub
Private Sub ConnectionTry(ByVal iar As IAsyncResult)
   .......
   _lblConnStatus.Invoke(New MethodInvoker(AddressOf successCon))
   .........
End Sub
```

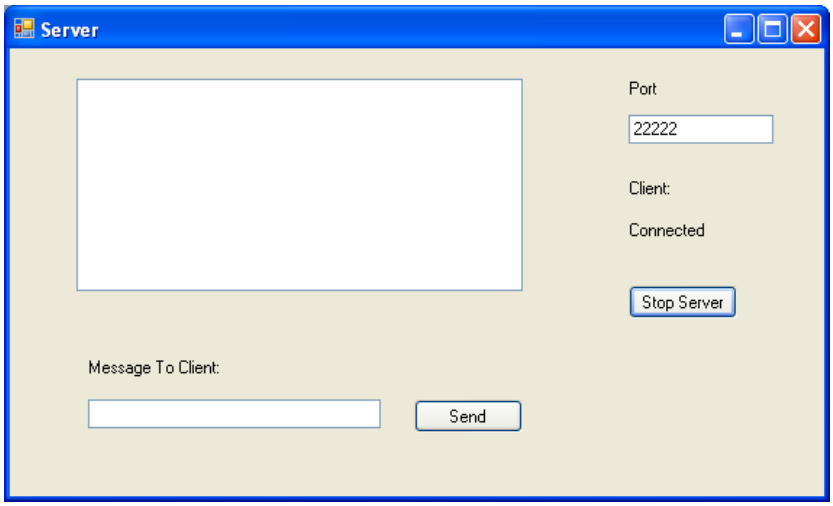

Fig.3 Interfața aplicației server, după pornirea serverului

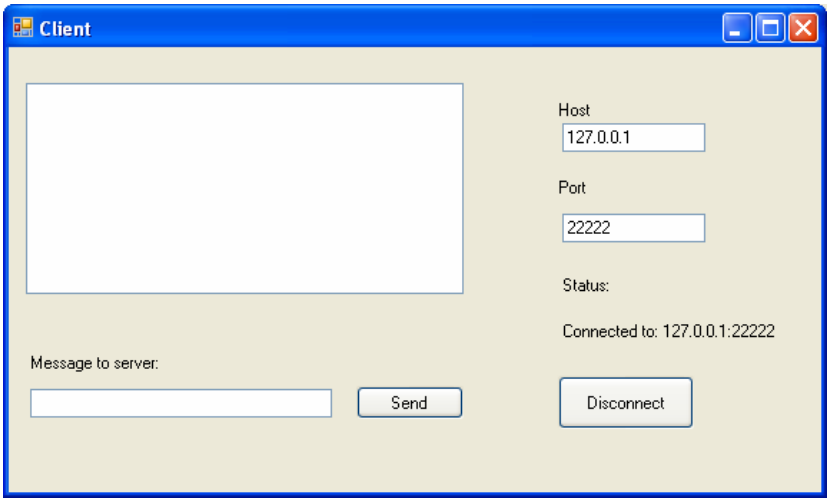

Fig.4 Interfața aplicației client, după conectarea la server

Câtă vreme, conexiunea este activă, se pot transmite şi recepționa date între client şi server. Conectarea în mod asincron la un calculator la distanță, oferă posibilitatea transmisiei şi recepției de date în cadrul unui fir de execuție separat

Pentru transmiterea datelor, se foloseşte metoda asincronă BeginSend, care începe o operație asincronă de transmisie a datelor, către un socket conectat.

Public Function BeginSend(ByVal *buffer***() As** Byte**, ByVal** *offset* **As** Integer**,**
ByVal *size* **As** Integer**, ByVal** *socketFlags* **As** SocketFlags**, ByVal** *callback* **As**
AsyncCallback**, ByVal** *state* **As** Object**) As** IAsyncResult

Parametrul *buffer* reprezintă un array de tip **Byte** care conține datele ce urmează să fie transmise. Parametrul *offset* reprezintă poziția în buffer de la care să se înceapă transmiterea datelor. Parametrul *size* reprezintă numărul de bytes de transmis. *Callback* reprezintă un delegate de tip **AsyncCallback** care va fi apelat când începe transmiterea datelor, şi se va executa într-un fir de execuție separat. Parametrul *state* reprezintă un obiect (orice fel de obiect) care conține informații despre starea cererii curente. Metoda returnează un obiect de tip **IAsyncResult** care se referă la transmisia asincronă efectuată.

Întrucât primul parametru cerut de metoda **BeginSend** este un array de tipul **Byte**, înainte de apelul metodei, trebuie facută conversia mesajului ce se doreşte a fi transmis din string într-un tablou de bytes.
Aceasta se realizează.cu ajutorul metodei GetBytes din clasa AsciiEncoding:

```
Dim message As Byte() = Encoding.ASCII.GetBytes(_txtBxMessages.Text)
```

Metoda BeginSend permite transmiterea datelor într-un fir de execuție separat. Se va crea o metodă de tip **AsyncCallback** care să fie dată ca parametru metodei **BeginSend**, pe care o vom numi **SendData.** Aceasta metodă callback va trebui să primească ca şi parametru , parametrul de tip IasyncResult returnat de metoda BeginSend şi să apeleze metoda **EndSend**, executându-se într-un fir de execuție separat.

Metoda **EndSend** încheie cu succes operația de transmisie asincronă începută de metoda **BeginSend**, returnand numarul de bytes transmisi.

Public Function EndSend (asyncResult As IAsyncResult) As Integer

Metoda **EndSend** va bloca firul de executie pana in momentul in care toate datele cerute vor fi transmise, sau pana cand este captata o eroare.

Pentru recepția datelor se folosește metoda asincronă BeginReceive, care începe o operație asincronă de recepție a datelor, de la un socket conectat.

Public Function BeginReceive (buffer As Byte(), offset As Integer, size As Integer, socketFlags As SocketFlags, callback As AsyncCallback, state As Object) As IAsyncResult

Parametrul *buffer* reprezintă un array de tip **Byte** în care se vor stoca datele recepționate. Parametrul *offset* reprezintă poziția în buffer de la care să se înceapă înscrierea datelor în buffer. Parametrul *size* reprezintă numărul de bytes primiți. *Callback* reprezintă un delegate de tip **AsyncCallback** care va fi apelat când începe receptia datelor, şi se va executa într-un fir de execuție separat. Parametrul *state* reprezintă un obiect (orice fel de obiect) care conține informații despre starea cererii curente. Metoda returnează un obiect de tip **IAsyncResult** care se referă la citirea asincronă efectuată.

Metoda **BeginReceive** face posibilă recepționarea datelor într-un fir de execuție separat. Se va crea o metodă de tip **AsyncCallback** care să fie dată ca parametru metodei **BeginReceive**, pe care o vom numi **ReceiveData.** Această metodă callback va trebui să primească ca şi parametru , parametrul de tip IasyncResult returnat de metoda BeginReceive şi să apeleze metoda **EndReceive**, executându-se într-un fir de execuție separat.

Metoda **EndReceive** încheie cu succes operația de citire asincronă începută de metoda **BeginRecieve**, returnand numarul de bytes cititi.

.

Public Function EndReceive (asyncResult As IAsyncResult) As Integer

```
Dim length As Integer = server.EndReceive(iar)
```

Metoda **EndReceive** va bloca firul de executie pana in momentul in care toate datele vor fi receptionate, sau pana cand este captata o eroare.

Întrucât datele recepționate sunt stocate într-un array de tipul Byte, ele vor trebui decodificate într-un string. Aceasta se realizează.cu ajutorul metodei GetString din clasa AsciiEncoding:

```
stringdata = Encoding.ASCII.GetString(buffer, 0, length)
```

Odată conversia facută, nu ne rămâne decât să afişăm mesajul primit în ListBox. Datorită faptului că ne aflăm într-o metodă de tip *delegate*, adăugarea unui *item* în ListBox se va face în felul următor:

```
Private Sub successReceive()
   _lstBxMessages.Items.Add("[Server] > " + stringData)
End Sub

Private Sub ReceiveData(ByVal iar As IAsyncResult)
   ...........
   _lstBxMessages.Invoke(New MethodInvoker(AddressOf
   successReceive))
   ...........
End Sub
```

Pentru închiderea unui socket trebuie apelată funcția **Close()** a instanței de tip Socket ce are ca efect închiderea conexiunii, dealocarea resurselor şi setarea proprietății Connected pe false.

În urma transmisiei şi recepției datelor, interfețele grafice ale celor două aplicații, vor arăta în felul următor:

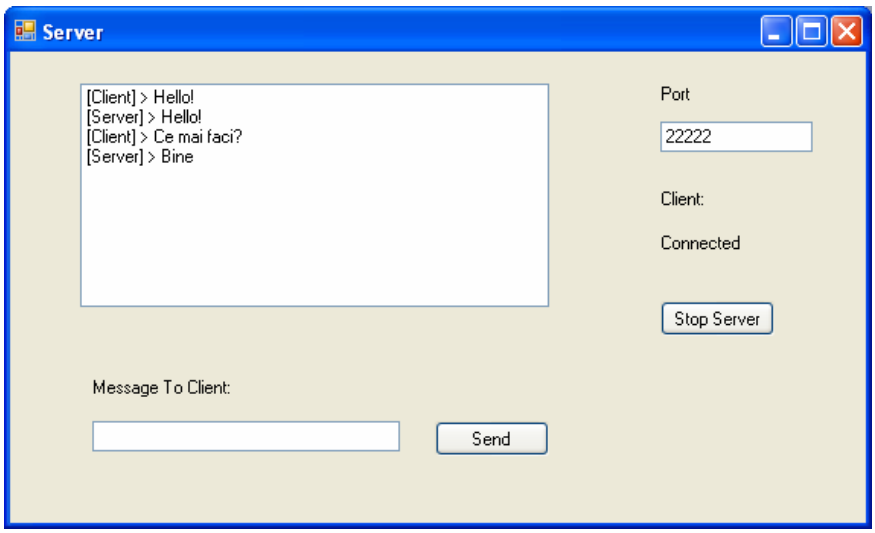

Fig.5 Interfața aplicației server, în timpul transmisiei şi recepției datelor

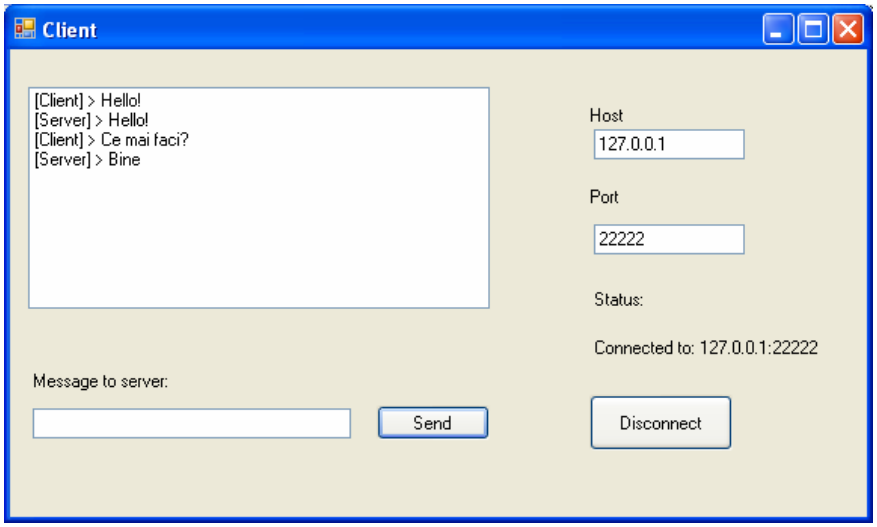

Fig.6 Interfața aplicației client, în timpul transmisiei şi recepției datelor

Obs. Se doreşte tratrea excepțiilor care pot să apară la pornirea serverului, la conectarea clientului la server, respectiv la transmisia şi recepția datelor între client şi server, utilizând . secvența:

```
try{
……
}
```

```
catch{
.....
}
```

În continuare vom spune câteva cuvinte despre o altă metodă de comunicare şi anume comunicarea sincronă (comunicare blocantă), care utilizează metodele Send si Receive pentru transmisia, respectiv recepția datelor. Atât metoda Send cât şi metoda Receive realizează blocaj. Aceasta înseamnă că apelul va fi blocat până când datele au fost trimise sau o excepție a fost aruncată, respectiv până când datele au fost recepționate sau o excepție a fost aruncată. Mai mult, de exemplu în cazul metodei Receive, chiar dacă sunt date când facem apelul, nu ştim când să facem apelul următor, şi din această cauză trebuie să facem verificări repetate (polling) dacă au venit sau nu date, rezolvare care nu este cea mai eficientă.

Pentru eliminarea acestor neajunsuri, se utilizează metoda de comunicare asincronă (comunicare neblocantă), pe care am prezentat-o şi in cadrul acestui laborator. Ea se caracterizează prin aceea că se porneşte un alt fir de execuție care se va bloca, aşteptând transmiterea sau recepția datelor şi va notifica firul principal de execuție de transmisia respectiv sosirea datelor. După cum s-a văzut, în acest mod de lucru s-au folosit metodele BeginSend si BeginReceive pentru transmisia, respectiv recepția datelor.